```
#ifndef GGCONSTANTS_H
#define GGCONSTANTS_H


const double ggFourPi = 12.566371;  /* needs more digits */
const double ggTwoPi = 6.2831853;  /* needs more digits */
const double ggPi = 3.14159265358979323846;
const double ggHalfPi = 1.57079632679489661923;
const double ggThirdPi = 1.0471976; /* needs more digits */
const double ggQuarterPi = 0.78539816; /* needs more digits */
const double ggInversePi = 0.31830989;
const double ggSqrtTwo = 1.41421356237309504880;
const double ggInverseSqrtTwo = 0.70710678;
const double ggSqrtThree = 1.7320508075688772935;
const double ggSqrtFive = 2.2360679774997896964;
const double ggE = 2.7182818284590452353602874;

const double ggRad = 57.29577951308232;

#ifdef sun
const double ggInfinity = 1.0e10;
#else
#include <float.h>
const double ggInfinity = DBL_MAX;
#endif

#ifndef M_PI
#define M_PI ggPi
#endif

const double ggBigEpsilon = 0.0001;
const double ggEpsilon = 0.000001;
const double ggSmallEpsilon = 0.000000001;
const double ggTinyEpsilon = 0.000000000001;


const double ggColorRatio = 0.00392156862745098803;

#endif

#define ggMin(x,y) (x < y) ? x : y
#define ggMax(x,y) (x > y) ? x : y
```

APPENDIX PAGE 1

APPENDIX PAGE 2

```
/* Includes required */
#ifdef WINDOWS
#include <windows.h>
#endif

#include <GL\gl.h>
#include <GL\glut.h>
#include <stdio.h>
#include <math.h>
#ifdef USE_NETPBM
#include <ppm.h>
#endif

/**
 * something because of windows.
 */
void __eprintf() {
}

/**
 * our data structure of choice
 */
typedef struct obj {
    /* other parameters */
    float matrix[16];

    /* view angle */
    float viewangle;

    /* aspect ratio */
    float aspect;

    /* z of the camera */
    float tz;

    /* ry of the camera */
    float ry;
} Obj;

/* hold the display lists for textures */
typedef struct texture {
    int tex1;
    int tex2;
} Texture;

/**
 * our global variables
 */
/* camera settings */
Obj scene;

/* texture stuff */
Texture def;
Texture* current_texture = &def;

/* track the next display list number */
int nextDLnum = 2;

/* stuff for lighting */
float lightPos[4] = {2.0, 4.0, 2.0, 0};
float lightDir[4] = {0, 0, 1.0, 1.0};
float lightAmb[4] = {0.3, 0.3, 0.3, 1.0};
float lightDiff[4] = {0.6, 0.6, 0.6, 1.0};
float lightSpec[4] = {0.6, 0.6, 0.6, 1.0};
float clipDistance = 2.14;
```

```
int left, right, parent;
int width, height;
#ifdef USE_NETPBM
pixel** ppmPixels = 0;
#endif
GLubyte* sgiPixels = 0;
FILE* commands;
int doTakeSnapshot = 0;

#define HEMISPHERE 1
void createHemisphere(int listNum, int numPts, int geom);
void draw_left();
void draw_right();
void Key(unsigned char,int,int);

/**
 * read the frame buffer and write out a ppm file
 */
void takeSnapshot() {
#ifdef USE_NETPBM
    static int shotNum = 0;
    FILE* file;
    char name[50];
    int index, i, j;

    /* draw everything again */
    draw_right();
    glFlush();
    draw_left();
    glFlush();

    /* read the pixels from the frame buffer */
    glReadPixels(0, 0, width, height, GL_RGB, GL_UNSIGNED_BYTE, sgiPixels);

    /* convert them to the ppm */
    index = 0;
    for (i = height - 1; i >= 0; i--) {
        for (j = 0; j < width; j++) {
            PPM_ASSIGN(ppmPixels[i][j],
                       sgiPixels[index],
                       sgiPixels[index + 1],
                       sgiPixels[index + 2]);
            index += 3;
        }
    }

    /* open a file */
    sprintf(name, "%d.ppm", shotNum);
    shotNum++;
    file = fopen(name, "w");

    /* write the ppm file */
    ppm_writeppm(file, ppmPixels, width, height, 255, 0);

    /* close the file */
    fclose(file);
#endif
}

/**
 * Read in the ppm files and create display lists for a texture
 * returns the dimension of the image
 */
void readTexture(Texture* t, char* file1, char* file2) {
#ifdef USE_NETPBM
```

APPENDIX PAGE 3

```
FILE *fp1, *fp2;
int cols, rows, i, j, index;
pixel **map1, **map2;
GLubyte *tex1, *tex2;
pixval maxval;

/* open the files */
fp1 = fopen(file1, "r");
fp2 = fopen(file2, "r");
if (!fp1) {
    fprintf(stderr, "Couldn't open %s\n", file1);
}
if (!fp2) {
    fprintf(stderr, "Couldn't open %s\n", file2);
}

/* read the ppm files */
map1 = ppm_readppm(fp1, &cols, &rows, &maxval);
fprintf(stderr, "%s: rows = %d \t cols = %d\n", file1, rows, cols, maxval);
map2 = ppm_readppm(fp2, &cols, &rows, &maxval);
fprintf(stderr, "%s: rows = %d \t cols = %d\n", file2, rows, cols, maxval);

/* convert them */
tex1 = malloc(sizeof(GLubyte) * rows * cols * 3);
tex2 = malloc(sizeof(GLubyte) * rows * cols * 3);
index = 0;
for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        /* R */
        tex1[index] = PPM_GETR(map1[i][j]);
        tex2[index] = PPM_GETR(map2[i][j]);
        index ++;

        /* G */
        tex1[index] = PPM_GETG(map1[i][j]);
        tex2[index] = PPM_GETG(map2[i][j]);
        index ++;

        /* B */
        tex1[index] = PPM_GETB(map1[i][j]);
        tex2[index] = PPM_GETB(map2[i][j]);
        index ++;
    }
}

/* create the textures in the left*/
glutSetWindow(left);
/* new display list*/
glNewList(nextDLnum, GL_COMPILE);
t->tex1 = nextDLnum;
nextDLnum++;
glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
            tex1);
glEndList();

/* new display list*/
glNewList(nextDLnum, GL_COMPILE);
t->tex2 = nextDLnum;
nextDLnum++;
glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
            tex2);
glEndList();

/* create the textures in the right*/
glutSetWindow(right);
```

```
    /* new display list*/
    glNewList(t->tex1, GL_COMPILE);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                tex1);
    glEndList();

    /* new display list*/
    glNewList(t->tex2, GL_COMPILE);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, cols, rows, 0, GL_RGB, GL_UNSIGNED_BYTE,
                tex2);
    glEndList();

    free(tex1);
    free(tex2);
    free(map1);
    free(map2);
#endif
}

/**
 * this will initialize the display lists for the objects
 */
void initialize_objects(int argc, char**argv) {
    float tmp[4];

    /* read in the texture */
    readTexture(&def, argv[1], argv[2]);

    /* create hemisphere left */
    glutSetWindow(left);
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* create hemisphere right */
    glutSetWindow(right);
    createHemisphere(1, 50, GL_TRIANGLE_STRIP);

    /* scene */
    scene.viewangle = 50;
    scene.tz = 0;
}


void display_parent() {
    /* clear the screen */
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

void draw_left() {
    float tmp[4];

    /* clear the screen */
    glutSetWindow(left);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* adjust for scene orientation */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(scene.viewangle, scene.aspect, 0.1, 10.0);
    glRotatef(scene.ry, 1, 0, 0);
    glTranslatef(0, 0, scene.tz);

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
```

APPENDIX PAGE 4

```
    /* now draw the semisphere */
    glEnable(GL_TEXTURE_2D);
    glColor3f(.5, .5, .5);
    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    glRotatef(180, 0.0, 0.0, 1.0);
    glColor3f(1, 1, 1);

    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();

    fprintf(stderr, "left - %s\n", gluErrorString(glGetError()));
}
void display_left()
{
    draw_left();
    glutSwapBuffers();
}

void draw_right() {
    float tmp[4];
    float height;

    /* clear the screen */
    glutSetWindow(right);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    /* setup the camera */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45, scene.aspect, clipDistance, 10.0);
    glTranslatef(0, 0, -3);
    glRotatef(15, 1, 0, 0);
    glRotatef(15, 0, 1, 0);
    glDisable(GL_TEXTURE_2D);

    /* draw our models */
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    /* draw a cube for the camera */
    glPushMatrix();
    glLoadIdentity();
    glRotatef(180, 1, 0, 0);
    glRotatef(-scene.ry, 1, 0, 0);
    glTranslatef(0, 0, scene.tz);

    tmp[0] = tmp[1] = tmp[2] = .8;
    tmp[3] = 1;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glutSolidCube(.1);

    /* now the light */
    tmp[0] = tmp[1] = tmp[2] = 0;
    tmp[3] = 1;
    glLightfv(GL_LIGHT1, GL_POSITION, tmp);
    glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, scene.viewangle / 2);
    tmp[0] = tmp[1] = 0; tmp[2] = 1; tmp[3] = 1;
    glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, tmp);

    /* draw a cone for the view frustrum */
    glLoadIdentity();
    height = 1 - scene.tz;
    glRotatef(-scene.ry, 1, 0, 0);
    glRotatef(45, 0, 0, 1);
    glTranslatef(0, 0, -1);
    tmp[0] = tmp[1] = .8;
    tmp[2] = 0;
    tmp[3] = 1.0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glutWireCone(tan(scene.viewangle * 3.14 / 360.0) * height, height, 10, 4);
    glPopMatrix();

    /* draw one half of the sphere */
    tmp[0] = tmp[1] = tmp[2] = tmp[3] = 0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tmp);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glEnable(GL_TEXTURE_2D);
    tmp[0] = tmp[1] = tmp[2] = tmp[3] = 1.0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, tmp);
    glCallList(current_texture->tex1);
    glCallList(HEMISPHERE);

    /* draw the other half */
    glRotatef(180, 0, 0, 1);
    glCallList(current_texture->tex2);
    glCallList(HEMISPHERE);
    glPopMatrix();
    fprintf(stderr, "right - %s\n", gluErrorString(glGetError()));
}

void display_right() {
    draw_right();
    glutSwapBuffers();
}

/*
 * Handle Menus
 */
#define M_QUIT 1
void Select(int value)
{
    switch (value) {
    case M_QUIT:
        exit(0);
        break;
    }
    glutPostRedisplay();
}
void create_menu() {
    glutCreateMenu(Select);
    glutAddMenuEntry("Quit", M_QUIT);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}


/* Initializes hading model */
void init_left()
{
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glEnable(GL_TEXTURE_2D);
```

APPENDIX PAGE 5

```
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
}

void init_right() {
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);

    /* texture stuff */
    glEnable(GL_TEXTURE_2D);
    glPixelStorei(GL_UNPACK_ALIGNMENT, sizeof(GLubyte));
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

    /* for blending */
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    /* lighting */
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiff);
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpec);

    glEnable(GL_LIGHT1);
    lightAmb[2] = 0;
    lightAmb[0] = lightAmb [1] = .8;
    lightAmb[3] = 1;
    glLightfv(GL_LIGHT1, GL_AMBIENT, lightAmb);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lightDiff);
    glLightfv(GL_LIGHT1, GL_SPECULAR, lightSpec);
}

/*
 *  Called when the window is first opened and whenever
 *  the window is reconfigured (moved or resized).
 */
void myReshape(int w, int h)
{
    /* set width and height */
    width = w;
    height = h;

    /*  define the viewport */
    glViewport (0, 0, w, h);
    scene.aspect = 1.0*(GLfloat)w/2/(GLfloat)h;

    /* reshape the subwindows */
    /* first left */
    glutSetWindow(left);
    glutReshapeWindow(w/2, h);

    /* now right */
    glutSetWindow(right);
    glutReshapeWindow(w/2, h);
    glutPositionWindow(w/2, 0);

    /* allocate memory for the snapshot thingy */
    if (sgiPixels) {
        free(sgiPixels);
    }
    sgiPixels = malloc(sizeof(GLubyte) * width * height * 3);

#ifdef USE_NETPBM
    if (ppmPixels) {
        ppm_freearray(ppmPixels, height);
    }
    ppmPixels = ppm_allocarray(width, height);
#endif
}

/**
 * an idle function to do automate things
 */
#define MOVE 0
#define SNAP 1
void idleSnapshots() {
    static moveOrSnap = MOVE;

    if (moveOrSnap == MOVE) {
        /* next command */
        Key('r', 0, 0);
        moveOrSnap = SNAP;
    } else {
        /* snapshot */
        Key(13, 0, 0);
        moveOrSnap = MOVE;
    }
}

/**
 * Rotate both displays
 */
void doRotate(float amt, float x, float y, float z) {
    glutSetWindow(left);
    glRotatef(amt, x, y, z);
    glutSetWindow(right);
    glRotatef(amt, x, y, z);
}

/*
 * Keyboard handler
 */
void
Key(unsigned char key, int x, int y)
{
    int keyInt;
    float matrix1[16];
    float matrix2[16];
    glutSetWindow(left);
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix1);
    glLoadIdentity();
    glutSetWindow(right);
    glMatrixMode(GL_MODELVIEW);
    glGetFloatv(GL_MODELVIEW_MATRIX, matrix2);
    glLoadIdentity();

    /* check for read command */
    if (key == 'r') {
        fscanf(commands, "%d", &keyInt);
        key = (char) keyInt;
        fprintf(stderr, "read %d - %c \n", key, key);
    }

    printf("%d\n", key);
    fflush(stdout);
    switch (key) {
    case '!':
        /* register idle function */
        glutSetWindow(parent);
        glutIdleFunc(idleSnapshots);
        break;
    case 13:
        /* press enter - take snapshot*/
        doTakeSnapshot = 1;
        break;
    case 'c':
        clipDistance -= .C2;
        break;
    case 'C':
        clipDistance += .02;
        break;
    case 'y':
        printf("ry = %f\n", scene.ry);
        scene.ry -= 5;
        break;
    case 'Y':
        scene.ry += 5;
        break;
    case 'z':
        scene.tz -= .C2;
        break;
    case 'Z':
        scene.tz += .02;
        break;
    case 'a':
        scene.viewangle -= 1;
        break;
    case 'A':
        scene.viewangle += 1;
        break;
    case 55:
        doRotate(-5, 0.0, 0.0, 1.0);
        break;
    case 57:
        doRotate(5, 0.0, 0.0, 1.0);
        break;
    case 52:
        doRotate(-5, 0.0, 1.0, 0.0);
        break;
    case 54:
        doRotate(5, 0.0, 1.0, 0.0);
        break;
    case 56:
        doRotate(5, 1.0, 0.0, 0.0);
        break;
    case 50:
        doRotate(-5, 1.0, 0.0, 0.0);
        break;
    case '?':
        fprintf(stderr, "arrows - rotate the sphere\n");
        fprintf(stderr, "a/A viewangle\n");
        fprintf(stderr, "c/C adjust clip plane in right window\n");
        fprintf(stderr, "z/Z camera position\n");
        fprintf(stderr, "Escape quits \n");
        break;
    case 27:            /* Esc will quit */
        exit(1);
        break;
    default:
        fprintf(stderr, "Unbound key - %d  \n", key);
        break;
    }
    fprintf(stderr, "clip = %f  viewangle = %f  zdepth = %f \n",
            clipDistance, scene.viewangle, scene.tz);
    glutSetWindow(left);
    glMultMatrixf(matrix1);
    glutPostRedisplay();
    glutSetWindow(right);
    glMultMatrixf(matrix2);
    glutPostRedisplay();

    /* check for snapshot */
    if (doTakeSnapshot) {
        takeSnapshot();
        doTakeSnapshot = 0;
    }
}



/*
 * Main Loop
 * Open window with initial window size, title bar,
 * RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA);

    /* create the parent window */
    parent = glutCreateWindow (argv[0]);
    width = 512; height = 256;
    glutReshapeWindow(width, height);
    glutKeyboardFunc(Key);
    glutReshapeFunc (myReshape);
    glutDisplayFunc(display_parent);
    create_menu();

    /* left window */
    left = glutCreateSubWindow (parent, 0, 0, width / 2, height);
    glutKeyboardFunc(Key);
    glutDisplayFunc(display_left);
    create_menu();
    init_left();

    /* right window */
    right = glutCreateSubWindow(parent, width / 2, 0, width / 2, height);
    glutKeyboardFunc(Key);
    glutDisplayFunc(display_right);
    create_menu();
    init_right();

    /* create objects */
    initialize_objects(argc, argv);

    /* open file */
    if (argc == 4) {
        fprintf(stderr, "Opening %s for commands\n", argv[3]);
        commands = fopen(argv[3], "r");
    }
    glutMainLoop();
}
```

APPENDIX PAGE 7

```
/*****************************/
/* warp.c                    */
/*****************************/

#include <math.h>
#include "ggConstants.h"

/*
 * This function takes a point in the unit square, and maps it to
 * a point on the unit hemisphere.
 *
 * Copyright 1994 Kenneth Chiu
 *
 * This code may be freely distributed and used for any purpose, commercial
 * or non-commercial as long as attribution is maintained.
 */
void
map(double x, double y, double *x_ret, double *y_ret, double *z_ret) {

    double xx, yy, offset, theta, phi;

    x = 2*x - 1;
    y = 2*y - 1;

    if (y > -x) {               /* Above y = -x */
        if (y < x) {                /* Below y = x */
            xx = x;
            if (y > 0) {                /* Above x-axis */

/*              cerr<<"Octant 1 ";*/

                offset = 0;
                yy = y;
            } else {                    /* Below and including x-axis */
/*                cerr<<"Octant 8 ";*/
                offset = (7*ggPi)/4;
                yy = x + y;
            }
        } else {                    /* Above and including y = x */
            xx = y;
            if (x > 0) {                /* Right of y-axis */

/*                cerr<<"Octant 2 ";*/

                offset = ggPi/4;
                yy = (y - x);
            } else {                    /* Left of and including y-axis */

/*                cerr<<"Octant 3 ";*/

                offset = (ggTwoPi)/4;
                yy = -x;
            }
        }
    } else {                    /* Below and including y = -x */
        if (y > x) {                /* Above y = x */
            xx = -x;
            if (y > 0) {                /* Above x-axis */

/*                cerr<<"Octant 4 ";*/

                offset = (3*ggPi)/4;
                yy = -x - y;
            } else {                    /* Below and including x-axis */

/*                cerr<<"Octant 5 ";*/

                offset = (ggFourPi)/4;
                yy = -y;
            }
        } else {                    /* Below and including y = x */
            xx = -y;
            if (x > 0) {                /* Right of y-axis */

/*                cerr<<"Octant 7 ";*/

                offset = (6*ggPi)/4;
                yy = x;
            } else {                    /* Left of and including y-axis */
                if (y != 0) {

/*                    cerr<<"Octant 6 ";*/

                    offset = (5*ggPi)/4;
                    yy = x - y;
                } else {

/*                    cerr<<"Origin ";*/

                    *x_ret = 0;
                    *y_ret = 1;
                    *z_ret = 0;
                    return;
                }
            }
        }
    }

    theta = acos(1 - xx*xx);
    phi = offset + (ggPi/4)*(yy/xx);

    *x_ret = sin(theta)*cos(phi);
    *y_ret = cos(theta);
    *z_ret = sin(theta)*sin(phi);
}

/*
 * This function takes a point in the unit hemisphere, and maps it to
 * a point on the unit square.
 *
 * Copyright 1994 Keneth Chiu and Kurt Zimmerman
 *
 *
 * This code may be freely distributed and used for any purpose, commercial
 * or non-commercial as long as attribution is maintained.
 */
void
unmap(double x, double y, double z, double *x_ret, double *y_ret)
{
        double xx, yy, offset, theta, phi;

        theta = acos(y);

        if(theta < .0000001)   /* vertical center */
        {
                *x_ret = 0.5;
                *y_ret = 0.5;
                return;
        }
        else
```

APPENDIX PAGE 8

```
{
    double cosphi,sinphi;
    cosphi = x/sin(theta);
    cosphi = ggMin(cosphi, 1.0); /* hack for now */
    cosphi = ggMax(cosphi, -1.0);
    sinphi = z/sin(theta);
    sinphi = ggMin(sinphi, 1.0); /* hack for now */
    sinphi = ggMax(sinphi, -1.0);

    if(sinphi >= 0 )
      phi = acos(cosphi);
    else if (sinphi < 0 && cosphi< 0)
      phi = -acos(cosphi);
    else
      phi = asin(sinphi);

    xx = sqrt(1 - y);

    if(phi < -(3 * M_PI)/4)
    {
        /* cerr<<"5th octant  ";*/
        yy = ((phi + M_PI) * xx)/(M_PI/4);
        *x_ret = -xx;
        *y_ret = -yy;
    }
    else if(phi <  -M_PI/2)
    {
        /*cerr<<"6th octant  ";*/
        yy = ((phi + (3*M_PI/4)) * xx)/(M_PI/4);
        *y_ret = -xx;
        *x_ret = *y_ret + yy;
    }
    else if(phi < -(M_PI)/4)
    {
        /*cerr<<"7th octant  ";*/
        yy = ((phi + M_PI/2) * xx)/(M_PI/4);
        *x_ret = yy;
        *y_ret = -xx;
    }
    else if(phi < 0)
    {
        /*cerr<<"8th octant  ";*/
        yy = ((phi + M_PI/4) * xx)/(M_PI/4);
        *x_ret = xx;
        *y_ret = yy - *x_ret;
    }
    else if(phi <  (M_PI)/4)
    {
        /*cerr<<"1st octant  ";*/
        yy = ((phi) * xx)/(M_PI/4);
        *x_ret = xx;
        *y_ret = yy;
    }
    else if(phi < M_PI/2)
    {
        /*cerr<<"2nd octant  ";*/
        yy = ((phi - M_PI/4) * xx)/(M_PI/4);
        *y_ret = xx;
        *x_ret = *y_ret - yy;
    }
    else if(phi < 3*M_PI/4)
    {
        /*cerr<<"3rd octant  ";*/
        yy = ((phi - M_PI/2) * xx)/(M_PI/4);
        *x_ret = -yy;
        *y_ret = xx;
    }
    else
    {
        /*cerr<<"4th octant  ";*/
        offset = 3*M_PI/4;
        yy = ((phi - offset) * xx)/(M_PI/4);
        *x_ret = -xx;
        *y_ret = -yy - *x_ret;
    }

}
*y_ret = (*y_ret + 1)/(2 + ggEpsilon);
*x_ret =(*x_ret + 1)/(2 + ggEpsilon);
}
```

```
/********************************/
/* warp.h                       */
/********************************/

#ifndef WARP_DOTH
#define WARP_DOTH

void map(double x, double y, double *x_ret, double *y_ret, double *z_ret);
void unmap(double x, double y, double z, double *x_ret, double *y_ret);


#endif
```